AI Magazine Volume 6 Number 3 (1985) (© AAAI)

Starting a Knowledge Engineering Project: A Step-by-Step Approach

Mike Freiling, Jim Alexander, Steve Messick, Steve Rehfuss, and Sherri Shulman

Artificial Intelligence Department, Computer Research Laboratory, Tektronix, Inc , Post Office Box 500, Beaverton, Oregon 97077

Getting started on a new knowledge engineering project is a difficult and challenging task, even for those who have done it before. For those who haven't, the task can often prove impossible. One reason is that the requirements-oriented methods and intuitions learned in the development of other types of software do not carry over well to the knowledge engineering task. Another reason is that methodologies for developing expert systems by extracting, representing, and manipulating an expert's knowledge have been slow in coming.

At Tektronix, we have been using a step-by-step approach to prototyping expert systems for over two years now. The primary features of this approach are that it gives software engineers who do not know knowledge engineering an easy place to start, and that it proceeds in a step-by-step fashion from initiation to implementation without inducing conceptual bottlenecks into the development process. This methodology has helped us collect the knowledge necessary to implement several prototype knowledge-based systems, including a troubleshooting assistant for the Tektronix FG-502 function generator and an operator's assistant for a wave solder machine.

One fundamental assumption we make is that knowledge is more valuable than inference strategies. Often a company may have only one chance to acquire the knowledge, but can work on it later at leisure. A second assumption is that a knowledge engineering project must provide adequate documentation of its progress. At any stage in the process, knowledge engineers must be able to show some fruits of their labor.

The Need for Knowledge Engineering Methodologies

In any large organization it is quite common to find "pockets of knowledge" or "knowledge bottlenecks." Pockets of knowledge occur when knowledge crucial to the success of an organization is possessed by only one or a few individuals. Knowledge bottlenecks are pockets of knowledge that impede an organization's progress because the knowledge needs to be more widely distributed.

For example, if knowledge about how to keep an important manufacturing process running smoothly resides in the head of only one or two process engineers, we have a pocket of knowledge. If the company now wants to build several similar plants in different international locations, we have a knowledge bottleneck, because the knowledge cannot be distributed as easily as can the material used to build a factory. The lore of manufacturing processes includes stories of engineers who were shuttled by plane between factories in an effort to keep them all running.

It is clear that knowledge pockets and bottlenecks are undesirable and should be eliminated if possible. Pockets of knowledge can quickly become serious bottlenecks if the individuals retire or decide to leave the organization.

Expert system technology has been offered as a means for removing knowledge pockets and bottlenecks. But despite some notable successes, the path to expert system implementation is fraught with difficulties. Among these difficulties are

- **The "AI Mystique."** Terms like "artificial intelligence" or "knowledge engineering" give the impression that there is something magical and/or mystical involved in building expert systems. Despite our claims about making it clear how everyone else does their job, we have had some difficulty making it clear how we do our own. As a result, knowledge engineering is often considered a technology that is far too difficult to attempt.
- **The management problem.** How is it possible to manage the progress of an expert system

FIGURE 1

Fragment of GLIB grammar
defining temporal
relations between signals

```
<temporal predicate> ::=
        <interval expression> BEFORE <interval expression> |
        <interval expression> DURING <interval expression> |
        <interval expression> AFTER  <interval expression> |
        <interval expression> WHENEVER <interval expression> |
        <ordinal integer> TIME THAT <interval expression> |
        <ordinal integer> OCCURRENCE OF <interval expression> |
        FROM <interval expression> UNTIL <interval expression>


<interval expression> ::=
        <interval adjustment> |
        <atomic interval expression>


<interval adjustment> ::=
        <interval expression> DELAYED BY <temporal value>


<atomic interval expression> ::=
        <signal> BECOMES <qualitative state> |
        <signal> ATTAINS <property> OF <value> |
        <signal> CROSSES <signal> |
        <signal> CROSSES <signal> GOING <polarity> |
        <signal> CROSSES <amplitude value> |
        <signal> CROSSES <amplitude value> GOING <polarity> |
        THE INTERVAL WHEN THE <property> OF <signal> <comparator> <value>

<polarity> ::= POSITIVE | NEGATIVE

<qualitative state> ::= HIGH | LOW  | ZERO | NEGATIVE

<property> ::= AMPLITUDE | FREQUENCY

<signal> ::= SIGNAL-<unsigned integer>

<value> ::= <amplitude value> | <frequency value>

<amplitude value> ::= <number> VOLTS

<frequency vlaue> ::= <number> HERTZ

<temporal value> ::= <number> SECONDS

<comparator> ::= IS | = | < | > | <= | >= | ~=
```

Figure 1.

project? The common wisdom is to build a prototype as quickly as possible. But what can be done to manage the project while the first prototype is being built?

- **Choosing the right tool.** A number of knowledge engineering tools are available on the market today. They range from simple backward chaining inference engines similar to EMYCIN (VanMelle, 1984) to sophisticated object-oriented environments that permit a number of different inference strategies to be implemented (Kunz, 1984). Assess the problem to be attacked before deciding which tool to use or whether a tool is needed at all
- **The acceptance problem.** How people will react to using knowledge-based consultants is a big question mark Even in the prototyping stages, it is important to plan systems with acceptance in mind and to build an acceptance-oriented interface for the first prototype

Several tools and methodologies have been developed to help manage the early phases of a knowledge engineering project. ETS is a knowledge acquisition system developed by Boeing Computer Services that acquires knowledge via a dialogue with the user and can actually build rule bases for several expert system tools (Boose, 1984).

Stefik *et al.* (1982) have articulated a variety of inference strategies, along with problem characteristics that dictate one choice or another. For instance, strategies which completely exhaust the possible answers and pursue a single line of reasoning are recommended only for problems where the number of possible solutions is small, the data is reliable, and the knowledge is also reliable. Complex search methods, such as opportunistic scheduling, are recommended when a single source of knowledge or line of reasoning is insufficient.

At a more abstract level, Clancey (1984) has examined the inference structure of many classification systems and articulated two different types of inference step in these problems.

The first type of step involves actions of abstraction and refinement which accomplish those parts of the problem-solving process that are fairly well-understood and automatic. Examples of abstraction steps include qualitative abstraction of numeric values, such as classifying a voltage of 4.67 to be "high" and generalization of a particular collection of symptoms into a relevant general class of patients, like "heavy smoker." Examples of refinement steps include selection of a particular component fault to account for some failure mode when the relevant faults can be exhaustively enumerated and checked, and the selection of some specific disease from the category of diseases known to be causing the patient's illness.

The second type of step involves heuristic associations which make intuitive leaps that cannot be deductively justified and may require reconsideration. Examples of heuristic associations include the association between symptom classes and disease categories that may be responsible, or between measured values in a circuit and failure modes that may be responsible.

In a similar vein, Brachman and Levesque (1982) have identified rules related to terminology and rules related to the problem and argued that separate inference strategies (or *process structure* in Clancey's terms) are needed for each.

The methodology we will discuss here is not intended to replace any of this previous work. Rather, our approach provides a step-by-step approach to building familiarity with a knowledge engineering problem that makes it possible to use techniques such as knowledge level analysis with a better understanding of what is involved in the problem.

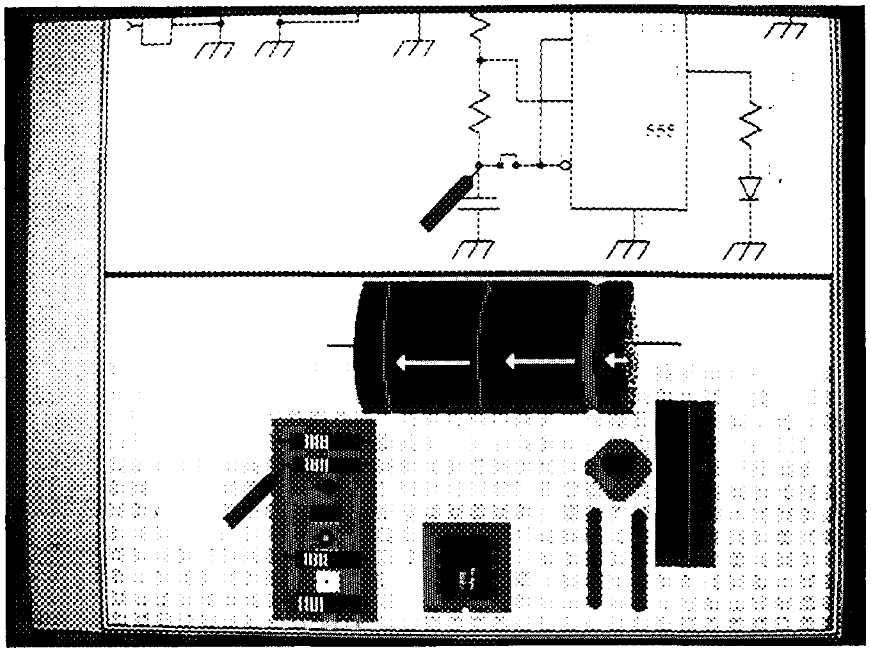
Requirements for a Knowledge Engineering Methodology

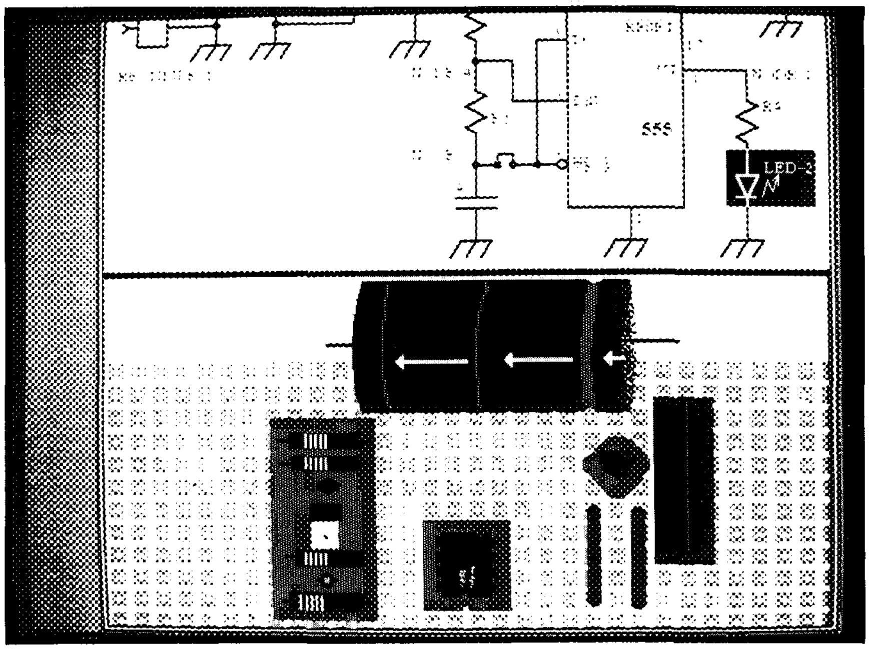
Before we get into answering the above question, let us step back a minute and ask, "What would we wish from a knowledge engineering methodology?"

1. *The methodology must be simple.* We all know knowledge engineering is a hard business, and there are many problems that exceed our ability right now. But there are also lots of small problems for which knowledge engineering in its present form is adequate. A methodology for attacking today's doable problems should be easy to apply and should lend itself to a wide variety of problems.
2 *The methodology must be gradual.* The people who need a knowledge engineering methodology most are those who do not have much prior experience with knowledge engineering Experience has shown that these people often encounter a conceptual bottleneck in attempting to formalize what initially appears to be an amorphous mass of knowledge.
3 *The methodology must aim at getting the knowledge first.* As we have seen, there are lots of cases where knowledge acquisition is the time-critical component. A methodology must help with this stage.
4. *The methodology must provide measurable milestones.* It is important to communicate a sense of progress in any project. If possible, there should be clear "deliverables" either on paper or in a working program to mark the progress towards a completed expert system.

Origins of the Step-by-Step Approach

Our approach had its start about two years ago when we began designing expert systems for troubleshooting Tektronix instruments. The project team consisted of an AI researcher, a cognitive psychologist, a software engineer, and an electronic engineer. The first problem we had to deal with was to establish a means of communication




555
LED-2

among such a diverse group. From work on the SIDUR project at Oregon State University (Freiling, 1983; Kogan, 1984), we had had some experience using formal grammars of English fragments as a documentation and communication tool. We decided to use a formal grammar to document the progress of our knowledge engineering efforts.

In order to build a grammar, however, we needed something to start with. So we sat down with several electronic engineers and technicians, and a schematic for the Tektronix FG-502 Function Generator, which we chose for its simplicity. A function generator is an instrument that generates waveforms of known shape and frequency as stimuli for testing electronic equipment. We asked these experts to tell us how they would go about troubleshooting the FG-502. We taped several hours of conversation and transcribed the troubleshooting knowledge onto paper. At this point, of course, the knowledge was in the form of English sentences.

As we collected more of the knowledge, we began to notice regularities. For example, the engineers would frequently mention the temporal relationship between two signals. The behavior of one signal would be considered important only when some other signal was at a particular value, such as low, or had already exhibited some event, such as crossing zero. Noticing that temporal comparisons were haphazardly scattered throughout many types of observations, we scheduled a couple of intensive sessions to define a systematic collection of comparisons between signals.

Gradually, our collection of example statements took the form of a grammar. During the transition, our document had a hybrid form which was part grammar and part examples. Finally, the grammar was at the point where we had enough to start the next phase of the knowledge engineering process. We christened the grammar GLIB (General Language for Instrument Behavior) (Freiling, 1984). Figure 1 shows a simplified fragment of the GLIB grammar for expressing the temporal relationships mentioned above. The semantics of these expressions are roughly the same as those used by other researchers in more general representations of time (Vilain, 1982; Allen, 1981).

At this stage, GLIB was not "complete" in any formal sense. GLIB is still undergoing modifications and extensions, as we learn new subtleties of the knowledge that electronic engineers and technicians possess. But the knowledge representation structures captured by GLIB at that point were sufficient to permit further progress.

We used GLIB as a guide to expressing the rules for our first prototype, the FG-502 troubleshooting assistant (Alexander, 1985). Although we did not then have a framework for rigorously enforcing GLIB syntax in our rule formats, the presence of the GLIB grammar greatly shortened the effort expended on acquiring actual rules, since we now had a collection of well-defined formats for expressing the knowledge acquired.

The primary focus of our prototype troubleshooting assistant for the FG-502 was on another aspect of the expert system implementation process, the development of a credible interface for technicians to use (Freiling, 1984a). As we mentioned earlier, the problem of acceptance of expert system technology requires serious consideration. From many discussions with technicians, we found that they were much more likely to be enthusiastic about expert systems if it offered them some personal added value.

Our clues to what this added value might consist of came by asking technicians what parts of their job were a needless waste of time. Several replied that leafing through the manual to find where a part is located was a distracting and frustrating task. Using these clues, we designed an interface for the FG-502 troubleshooting assistant which eliminates the need to consult a manual for part locations during the troubleshooting and repair process. Our use of Smalltalk (Goldberg, 1983) with its rich environment of graphics primitives, made it possible to implement this interface in a matter of weeks.

Using the interface, technicians can point to the location of a part in either a parts list, a schematic diagram, or a map of the actual circuit board, and retrieve the location and parts data automatically. This interface is also a great help during the troubleshooting process itself. Nodes which must be measured are indicated using icons that represent an oscilloscope probe, and parts to be removed are highlighted by reversing their color. Figures 2 and 3 show examples of this interface in use. The use of this type of interface, even (or perhaps, especially) in the first prototype, can have a major impact on the acceptance issues that every knowledge engineering project must face.

In helping others build their own expert system applications, we became aware of the wider applicability of this step-by-step approach to developing an expert system prototype. Communicating this approach to others has helped to minimize both development times and the level of external consulting required by other projects within Tektronix.

Our experience in developing these prototypes has encouraged us to build tools that support this approach to developing expert systems. One such tool is INKA (INglish Knowledge Acquisition), a knowledge acquisition system that uses the GLIB grammar to produce a parser that captures specific troubleshooting facts and rules (Phillips, 1985). We will discuss INKA in more detail later.

Steps to an Expert System Prototype

The overriding goal of our approach has been to reduce the costs associated with expert system development. We have called this approach the DETEKTR (Development Environment for TEKtronix TRoubleshooters) methodology, after a development environment (DETEKTR) we

KNOWLEDGE DEFINITION PHASE

NUMBER	STEP	PROJECT DOCUMENT
(1)	Familiarization	Paper knowledge base
(2)	Organizing knowledge	Knowledge acquisition grammar
(3)	Representing knowledge	Internal knowledge base formats

PROTOTYPE IMPLEMENTATION PHASE

NUMBER	STEP	PROJECT DOCUMENT
(4)	Acquiring knowledge	Knowledge base
(5)	Inference strategy design	Inference engine
(6)	Interface design	Interface

Table 1.

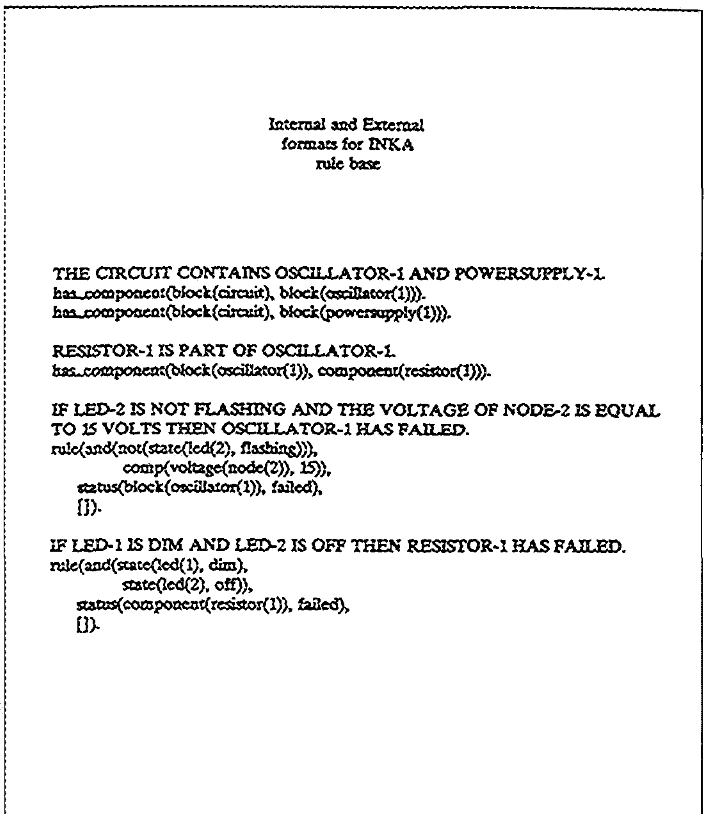
```
RULE-BL-4
IF      (1) V(N20) is not equal to V(N24).
THEN            (1) the source follower block is FAILED.

        EXPLANATION: Under correct behavior, a triangle voltage is being
        generated at N20 and N24, in phase with each other. Disruption of
        the loop causes voltages to converge to DC values, because the
        loop itself is required for alternating behavior.

RULE-BL-5
IF      (1) V(N24) is HIGH,
   AND  (2) V(N20) is HIGH,
   AND  (3) V(N19) is HIGH.
THEN            (1) the triangle wave comparator block is FAILED.

        EXPLANATION: Under correct behavior, the voltage at N19 acts as
        a "terminating condition" for the ramp being generated. Normally,
        then, it should be opposite in sign from the voltage at N20. When
        this does not occur, the comparator block is not generating the
        correct voltage at N19.

RULE-BL-6
IF      (1) V(N24) is LOW,
   AND  (2) V(N20) is LOW,
   AND  (3) V(N19) is LOW.
THEN            (1) the triangle wave comparator block is FAILED.

        EXPLANATION: All three of rules BL-3, BL-5, and BL-6 are
        needed to deduce failure in the triangle wave comparator
        block, because all three conditions are manifestations of
        various failure modes for that block.
```

Figure 4.

have been building to support it. We originally envisioned DETEKTR as a collection of domain-specific tools for building troubleshooters (Alexander, 1985a) because the costs of developing the grammar and interface could be amortized over a large number of troubleshooting assistants. But as we began to use our tools to support projects around Tektronix, we discovered that aspects of the methodology are relevant to almost any knowledge engineering problem.

Our methodology is divided into six steps, which fall into two general phases of the knowledge engineering project. The first phase is aimed at acquiring and representing the knowledge necessary for solving the problem. The second phase is aimed at actually constructing a prototype expert system.

Each step in the process also has an associated project document that forms a "deliverable" to mark successful conclusion of the step. This is important from the standpoint of managing a knowledge engineering project, since it is possible to demonstrate progress by means of these documents, even before programs are written.



```
Internal and External
formats for INKA
rule base

THE CIRCUIT CONTAINS OSCILLATOR-1 AND POWERSUPPLY-1.
has_component(block(circuit), block(oscillator(1))).
has_component(block(circuit), block(powersupply(1))).

RESISTOR-1 IS PART OF OSCILLATOR-1.
has_component(block(oscillator(1)), component(resistor(1))).

IF LED-2 IS NOT FLASHING AND THE VOLTAGE OF NODE-2 IS EQUAL
TO 15 VOLTS THEN OSCILLATOR-1 HAS FAILED.
rule(and(not(state(led(2), flashing))),
         comp(voltage(node(2)), 15)),
   status(block(oscillator(1)), failed),
   []).

IF LED-1 IS DIM AND LED-2 IS OFF THEN RESISTOR-1 HAS FAILED.
rule(and(state(led(1), dim),
         state(led(2), off)),
   status(component(resistor(1)), failed),
   []).
```

Figure 5.

The Knowledge Definition Phase

This is a phase of analysis and definition of the knowledge structures that precedes actual acquisition of knowledge and implementation of a prototype. The emphasis at this stage is to make progress on decomposing a large and complex problem, while not getting bogged down in the specifics of the problem.

Step 1: Familiarization. The purpose of the first, exploratory stage of a knowledge engineering project is to determine the scope and complexity of the task. Experienced knowledge engineers often know how to initiate this process by a combination of relatively unstructured interviews and observation sessions. Aspiring knowledge engineers, however, often get stymied at this point.

We have found it helpful to add structure to the initial interviews. The first thing to do is to pick a sample problem to work on that is more or less representative of the task for which you wish to build an expert system. It helps if this first example is on the simpler end of the complexity scale. It is possible to conduct the first sessions by either watching an expert solve the problem or talking to the expert about how the problem is most easily solved. Although experienced knowledge engineers might prefer the former approach because it is less intrusive, the more verbal approach is usually easier to start with.

A record of the sessions are made by taping them or by taking notes. After the session is over, the tape and the notes should be combed to produce a "paper knowledge base" consisting of English sentences that are representative expressions of the facts and rules the expert has given you. They do not, of course, need to be direct quotes from the expert. But they do need to be sentences that the expert can recognize as clear and unambiguous descriptions of relevant knowledge.

Figure 4 shows a fragment of the actual paper knowledge base produced in developing the FG-502 troubleshooting assistant. Notice that the paper knowledge base may employ highly stylized formats to make the knowledge structures clear. Also notice that at this stage of the knowledge engineering process, it is useful to store explanations for every single rule.

It may require many sessions with the expert to review and clarify the paper knowledge base before it reaches a stable form. The documentation of this stage of the project is the paper knowledge base itself.

Step 2: Organizing Knowledge. As the paper knowledge base collected in step 1 gets larger, it becomes unwieldy. At the same time, it should begin to exhibit some regularity in the sense that expressions of similar form reappear frequently in the document. The next step is to capture these regularities by building a knowledge acquisition grammar to express the facts and rules in the paper knowledge base. We are not suggesting that the grammar must be built along linguistic lines, which would require the use of grammatical categories like "prepositional phrase" and "relative clause." It is easier to build a "semantic grammar," (Burton, 1976) especially if the personnel involved do not have a background in linguistics. In a semantic grammar, the grammatical categories are derived from concepts related directly to the problem under consideration. For example, the GLIB syntax for the category ⟨ rule⟩ is

⟨rule⟩ ::= IF ⟨observation⟩ THEN ⟨conclusion⟩

and one grammar rule defining the category ⟨observation⟩ is

⟨observation⟩ ::= ⟨signal⟩ HAS ⟨property⟩ OF ⟨value⟩

Figure 1 shows a fragment of the syntax of GLIB. The documentation at this step of the project is the syntactic definition of this knowledge acquisition grammar.

Step 3: Representing Knowledge.

Once the English-like knowledge acquisition grammar has been specified, it is then used to guide the process of deciding how the knowledge is to be represented in a prototype expert system. The simplest way to do this is to begin with the categories of the semantic grammar. Nearly all these categories will be meaningful from the standpoint of a representation. It is necessary to determine a specific form for storing instances of the category. Assuming the target inference engine will run in Prolog, a corresponding syntax for the rule about observations might be

⟨observation⟩ ::= ⟨property⟩(⟨signal⟩ , ⟨value⟩)

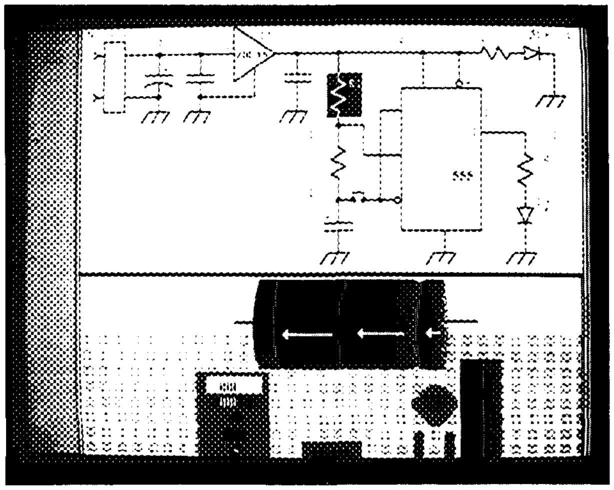
which produces structures like

voltage (signal-3, high)
frequency (signal-3, 5000)

The documentation at this stage the definition of internal knowledge base formats as they relate to the acquisition grammar syntax. We have used lexical functional grammar constraints (Kaplan, 1982) to accomplish this, but any appropriate mapping technique will suffice. Lexical functional grammar constraints will be discussed later. Figure 11 shows an example of these constraints, attached to a fragment of the GLIB grammar.

The Prototype Implementation Phase

Once the external and internal knowledge base formats have been defined, they can be used to guide the implementation of a prototype expert system. The implementation process consists of acquiring the knowledge base, building an inference engine, and building an appropriate interface.



Sample circuit display in INKA.

Figure 6.

Step 4: Acquiring Knowledge. Once a semantic grammar and a mapping to internal rule formats have been defined, it is possible to make a wholesale effort to acquire knowledge relevant to a particular task. This can be accomplished in several ways. The most convenient is to use a tool that allows the expert to generate English expressions conforming to the knowledge acquisition grammar and translates these automatically into the target formats.

Figure 5 shows some sample external and internal rules captured by the INKA knowledge acquisition tool for troubleshooting the simple circuit shown in Figure 6. We will discuss INKA in more detail later.

The documentation of this step is a prototype knowledge base, containing facts and rules specifically relevant to the prototype under construction. The prototype knowledge base will exist in two forms, an external knowledge base consisting of rules as acquired from the expert in English and an internal knowledge base, ready to be processed by some inference engine.

Step 5: Inference Strategy Design. Once a partial knowledge base has been acquired, it is time to build or select an inference engine to process the knowledge base. This is the point at which the work by Stefik *et al.* (1982) becomes relevant. Because of their prior exercises in acquiring and building the knowledge base, the project team has a much better familiarity with the requirements of the problem and are able to make a more educated choice of inference strategies. Ideally, choices about the knowledge representation would enlighten this decision, but to date we have not found any method to improve on Stefik's informal analysis. We have found in some simple cases, however, that the same knowledge base can be reused with progressively more sophisticated inference engines.

These findings partially confirm our hypothesis that extracting and articulating the knowledge is the most important phase of the expert system development process. Figure 7 shows simplified Prolog code for two different inference strategies for troubleshooting an electronic instrument. The first is called top-down localization and is similar to the engine used in INKA. The second uses a strategy of direct hypotheses to make educated guesses about where to look for failures. Although additional knowledge (in the form of heuristics) is required to make the hypothesis-based engine work, the actual tests are determined by the same knowledge as used by the first strategy.

The documentation at this stage of the project is a running inference engine.

Step 6: Interface Design. As we mentioned before, the design of an effective interface is extremely important in delivering acceptable expert systems. Generally this involves trying to discover what parts of the task are routine and can be handled in an effective interface. In the FG-502 troubleshooting assistant, for instance, we found that helping the technicians locate parts could save them time in troubleshooting instruments. Similar techniques have been used in managing graphical service documents at Brown (Feiner, 1982) and also on the Steamer project at BBN (Hollan, 1984). The number and quality of available primitives can have a large effect on the quality of an interface that can be produced in some fixed period of time. The documentation at this point in the project is a prototype interface.

Two troubleshooting inference machines that share knowledge

```
TOP-DOWN LOCALIZATION:
A SIMPLE INFERENCE MECHANISM

RULE FORMAT:
        diagnostic_rule_exists_for(Module, Unit).

PRIMARY INFERENCE LOOP:

        diagnose(Module, FailingUnit) :-
                is_replaceable_unit(Module),
                Module = FailingUnit.

                        ; terminate search when search leads to
                        ; a failing replaceable unit.

        diagnose(Module, FailingUnit) :-
                choose_subsystem(Module, SubUnit),
                diagnostic_rule_exists_for(SubUnit, Test),
                performs_successfully(Test),
                diagnose(SubUnit, FailingUnit).

HYPOTHESIS-BASED STRATEGY:
A MORE POWERFUL MECHANISM

RULE FORMATS:
        diagnostic_rule_exists_for(Module, Unit).

                        ; identical to previous rule,
                        ; same interpretation as before.

        heuristic_connection(Module, Observation, Suspect).

PRIMARY INFERENCE LOOP:

        diagnose(Module, FailingUnit) :-
                is_replaceable_unit(Module).
                Module = FailingUnit.

                        ; terminate search when search leads to
                        ; a failing replaceable unit.

        diagnose(Module, FailingUnit) :-
                heuristic_connection(Module, Observation, Suspect),
                performs_successfully(Observation),
                diagnose_hypothesis(Module, Suspect, FailingUnit).

                        ; this rule tries to find hypotheses for
                        ; a failure if no hypotheses are available.
                        ; failures occuring at a particular module.

        diagnose(Module, FailingUnit) :-
                choose_subsystem(Module, SubUnit),
                not(tested(SubUnit)),
                diagnostic_rule_exists_for(SubUnit, Test),
                performs_successfully(Test),
                diagnose(SubUnit, FailingUnit).

                        ; this rule provides a backup when no
                        ; hypotheses are applicable, that performs
                        ; identically to top-down localization.


        diagnose_hypothesis(Module, Suspect, FailingUnit) :-
                assert(tested(Suspect))
                diagnose(Suspect, FailingUnit).

                        ; this rule establishes that a suspect
                        ; has been tested (so it is only tested once)
                        ; and attempts to diagnose a failure in the
                        ; suspected module.
```

Two troubleshooting inference machines.

Figure 7.

Features of the Methodology

As we mentioned before, one major advantage of a methodological approach to knowledge engineering is that it provides a basis for communicating about the progress of a knowledge engineering project. All too often, knowledge engineering projects become a black hole, and managers have difficulty perceiving signs of progress. With a clear sense of stages and documentation which can be delivered at each milestone, it becomes possible to say "We've completed the knowledge organization phase and we're now defining the representation," rather than "We're working on it."

The major question that arises about our methodology is whether it is possible to define the forms for representing knowledge before understanding the uses to which the knowledge will be put. It has been observed by many researchers that knowledge engineering is very much a chicken and egg phenomenon (Buchanan, 1983). The types of knowledge that will be useful and the forms in which this knowledge should be represented depend on the inference strategies to process the knowledge. On the other hand, determining an appropriate inference strategy requires knowing something about the knowledge required to solve the problem.

How can this loop be broken? Our experience is that the knowledge as elicited from the expert consists primarily of references to objects, relations, observations, and events which are well known in the problem world. Only at the final level of grammatical processing do these references aggregate into heuristic connections, say between an observation and a conclusion. The part of GLIB that relates to specific assumptions about the inference strategies to be used, for instance, amounts to probably less than 1% of the total.

This knowledge is likely to contain much in the way of qualitative descriptions and categories which the expert finds useful in discussing the problem. To a first approximation, it is precisely these hints as to the correct levels of abstraction that we want to acquire from the expert in the first place.

If an expert instrument troubleshooter uses a rough qualitative characterization of voltage levels, such as "high" and "low" in describing how to troubleshoot the device, this qualitative categorization of the rules represents an abstraction that the expert finds useful. Whatever inference strategies we may utilize to heuristically connect behavior and its interpretation, it is quite likely that the underlying knowledge will be most effectively processed at this level of abstraction.

Of course, the problem still remains as to how to characterize the qualitative distinctions between "high" and "low" when measurements are made that do not automatically correspond to the expert's categorization. Brown *et al.* (1982) have discussed techniques for performing this particular qualitative abstraction. We are currently working on an analytical technique for defining these abstractions and their effects in any knowledge engineering problem.

With respect to the interpretation of specific heuristic associations, we note that these connections tend to appear only at the highest level of aggregation of an expert's knowledge. For example, particular troubleshooting rules might be captured from an expert in the form

IF ⟨observation⟩ THEN ⟨conclusion⟩

These rules might be interpreted as defining rigorous tests that guarantee the conclusion or heuristic associations that relate symptoms to their causes. Distinguishing between these possible interpretations is not possible on the basis of the syntactic expression alone.

In either case, the syntactic structure shown above is likely to be used. It is therefore possible to build the syntax for an expert's expressions without needing immediately to assign an interpretation to them. Herein lies the usefulness of having an external knowledge base, because these associations can even be captured without specific commitment to their interpretation in a particular inference strategy. When interpretation finally becomes inevitable, the external knowledge base provides a store of expressions for processing under whatever assumptions about interpretation are most appropriate. Even when changes are made to these interpretations, the knowledge need not be re-acquired. Even in extreme cases where a dialogue with the expert is necessary to make the correct interpretation, the syntactic structures of the external rule base provide a convenient means for discriminating those expressions that will require close analysis from those that will not.

```
              * * * * * * * * * * * * * * * * * *
                     WAVESOLDER ASSISTANT
                             MENU
              * * * * * * * * * * * * * * * * * *

          a. Insufficient solder
          b. Voids
          c. Solder bridges
          d. Icicles
          e. Peeling solder resist
          f. Nonwetting
          g. Warping
          h. Rosin film
          i. Removed Print
          j. Raised components
          k. Solder balls
          l. Missed joints
          m. Front edge of board poorly soldered
          n. Rough Solder
          q. QUIT

SELECT YOUR PROBLEM (a,b,c,...) d SHOW PROBLEM
DEFINITION (y/n) y

ICICLES: spike, flag, cone-shaped peak or sharp point of sol-
der

Return to menu? (y/n) n Is the solder temperature lower
than 450 degrees? (y,n) n Is the flux density higher than 850?
(y/n) n Is the preheat temperature lower than 730 degrees?
(y/n) y Increase the preheat temperature to 730 degrees.
```

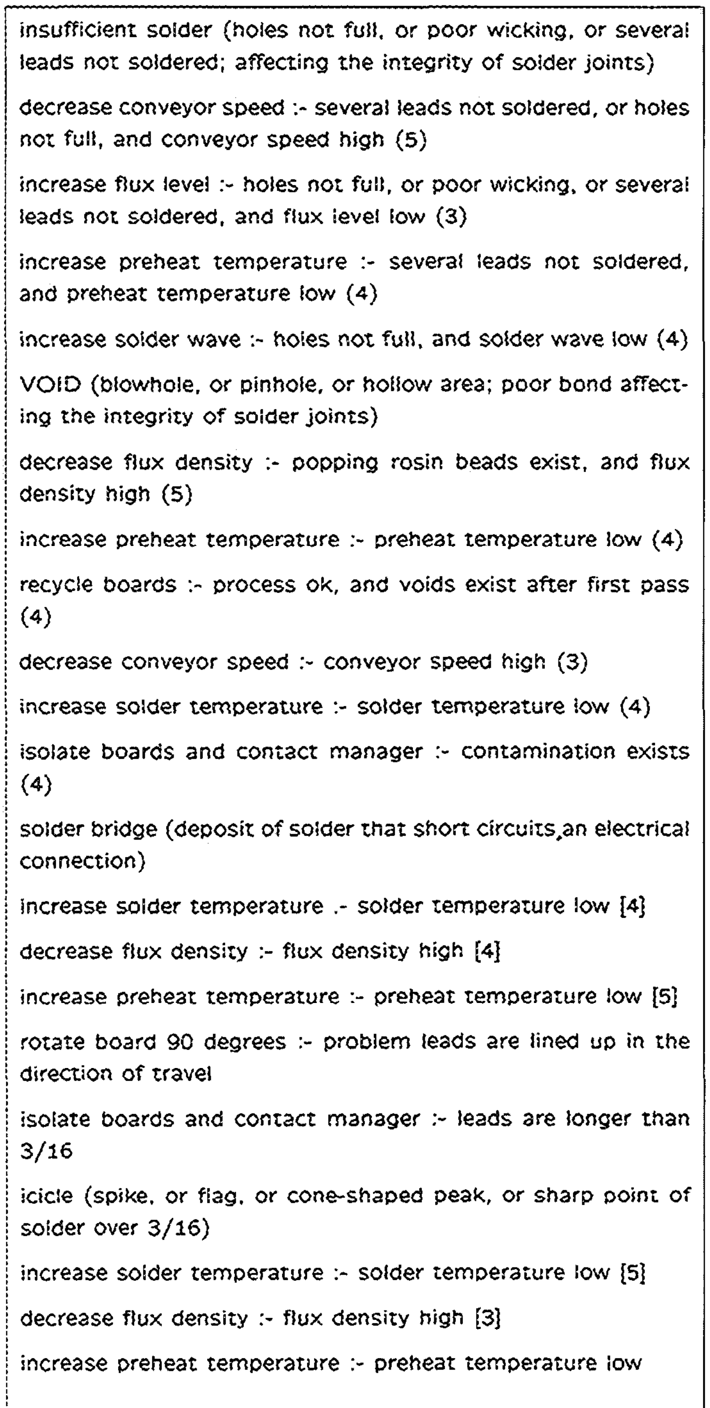
Sample Dialogue with Wave Solder Machine Operator's Advisor

Figure 8.

Variations on the Methodology

The methodology we have presented is, in fact, quite detailed. There are many examples of simple knowledge engineering applications where it does not make sense to follow this approach rigidly. Steps can be skipped or combined, depending on the common sense of the project team and its management.

An example of how this methodology may be varied to suit particular needs is provided by the history of development of a Wave Solder Machine Operator's Assistant, being built by Sal Faruqui and Bill Barton in Tektronix' Lab Instruments Division.



insufficient solder (holes not full, or poor wicking, or several leads not soldered; affecting the integrity of solder joints)

decrease conveyor speed :- several leads not soldered, or holes not full, and conveyor speed high (5)

increase flux level :- holes not full, or poor wicking, or several leads not soldered, and flux level low (3)

increase preheat temperature :- several leads not soldered, and preheat temperature low (4)

increase solder wave :- holes not full, and solder wave low (4)

VOID (blowhole, or pinhole, or hollow area; poor bond affecting the integrity of solder joints)

decrease flux density :- popping rosin beads exist, and flux density high (5)

increase preheat temperature :- preheat temperature low (4)

recycle boards :- process ok, and voids exist after first pass (4)

decrease conveyor speed :- conveyor speed high (3)

increase solder temperature :- solder temperature low (4)

isolate boards and contact manager :- contamination exists (4)

solder bridge (deposit of solder that short circuits,an electrical connection)

increase solder temperature .- solder temperature low [4]

decrease flux density :- flux density high [4]

increase preheat temperature :- preheat temperature low [5]

rotate board 90 degrees :- problem leads are lined up in the direction of travel

isolate boards and contact manager :- leads are longer than 3/16

icicle (spike, or flag, or cone-shaped peak, or sharp point of solder over 3/16)

increase solder temperature :- solder temperature low [5]

decrease flux density :- flux density high [3]

increase preheat temperature :- preheat temperature low

Wave Solder rule base in transit to Prolog clauses.

Figure 9.

Figure 8 shows a sample dialogue with the program. This program is not especially complicated. The wave solder machine operator checks boards as they are soldered for obvious defects like bridges or holes. When a defect is noticed, the operator must adjust one or more operating parameters of the soldering machine.

The important thing about the wave solder operator's assistant is that, though simple, it is typical of many small problems that can be profitably attacked with a knowledge engineering solution, provided costs can be kept acceptably low.

A variation on the DETEKTR methodology was used to manage development of this system. Because of the simplicity of the problem and the fact that it was to be a one-of-a-kind system, it was not necessary to formally define a knowledge acquisition grammar. Instead, the paper knowledge base was transcribed directly to Prolog.

Figure 9 shows an example of the paper knowledge base as it migrated into Prolog. The wave solder machine operator's assistant has passed preliminary testing by an expert on the solder machine's operation. Operational testing is scheduled for June, 1985.

Tools for Knowledge Engineering

The DETEKTR methodology provides a view of the knowledge engineering process that emphasizes acquisition and analysis of the knowledge prior to construction of a prototype. The inference strategy to be employed is considered only after much work has already been done.

This view dictates a concrete approach to the idea of a development environment for expert systems. Inference mechanisms are only one of the tools needed to support the progress from interview transcripts to prototype. The knowledge engineering environment to support a methodology like this will also need tools to:

- support the mapping from paper knowledge base to grammar.
- support analysis of the grammar and definition of the internal knowledge base formats.
- permit selection from a catalog of inference strategies.
- build natural language and graphical interfaces.

DETEKTR is designed as a prototype development environment for expert systems that consists of a collection of tools of the form described. These tools have been specialized in our case to support development of expert systems for troubleshooting electronic instruments. The principles behind the tools, however, apply to other problem worlds as well.

INKA is a tool that supports the acquisition of troubleshooting rules for electronic instruments. Using the INGLISH interface (Phillips, 1984), INKA translates expressions from GLIB into Prolog clauses, which are processed

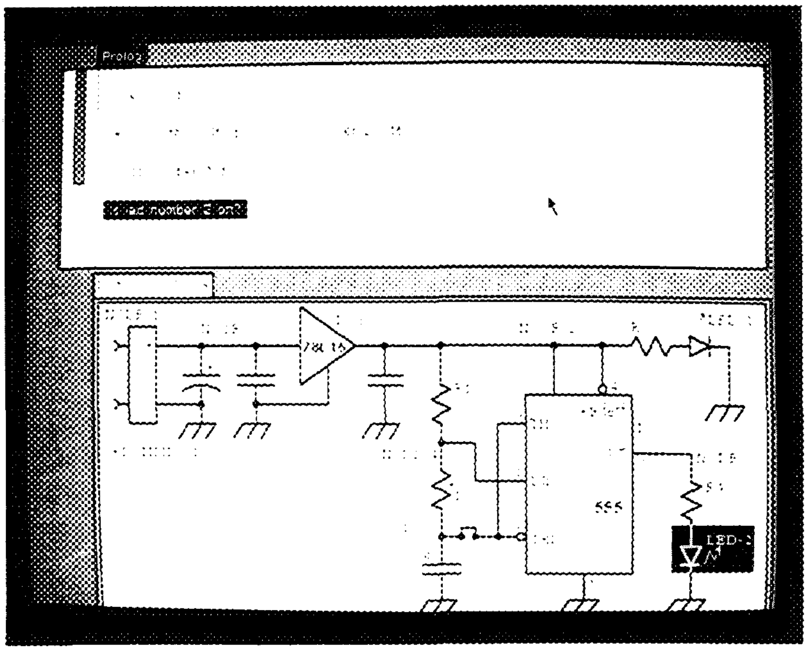


Figure 10

Fragment of GLIB grammar with LFG constraints

```
                                       (↑ CNCL)=↓
                  (↑ COND)=↓           (↑ FORM)=<rule((↑ COND FORM), (↑ CNCL FORM))>
<rule>      --> IF <condition> THEN <conclusion>

                                (↑ FORM)=<state((↑ IND), (↑ STATE))>
                 (↑ IND)=↓      (↑ STATE)=↓
<condition> --> <indicator> IS <state>

                 (↑ FORM)=<status((↑ DEV), failed)>
                 (↑ DEV)=↓
<conclusion>--> <device> HAS FAILED
```

Figure 11

by a specialized troubleshooting inference engine written in Prolog. INKA could easily be modified to provide a knowledge acquisition system for other problems as well. The input to INKA is a semantic grammar representing the external knowledge base format, and a mapping to an internal format. INKA acquires troubleshooting rules in the external format and passes them to a prolog inference engine for processing. Figure 10 shows INKA in operation.

The mapping to an internal format is accomplished by supplying lexical functional constraints (Kaplan, 1982) that map categories from the GLIB grammar to internal forms.

Figure 11 shows a fragment of the GLIB grammar that has been annotated with the necessary lexical functional constraints.

When a rule has been parsed, as shown in Figure 12, the constraints act to propagate functional attributes, like FORM, STATE, and IND, from lower grammatical categories to higher ones. Referring to Figure 12, for instance, the constraint above the category ⟨condition⟩,

$$(\uparrow \text{COND}) = \downarrow$$

governs propagation of all functional attributes (including one called FORM) to the level of the category ⟨rule⟩, to be stored under the functional attribute COND In like manner, the first constraint above ⟨conclusion⟩,

$$(\uparrow \text{CNCL}) = \downarrow$$

governs propagation of all attributes to the CNCL property of the ⟨rule⟩ category. Then the second constraint above ⟨conclusion⟩,

$$(\text{FORM}) = \langle \text{rule}((\uparrow \text{COND FORM}), (\uparrow \text{CNCL FORM}))\rangle$$

retrieves these two attributes, COND and CNCL, and combines their FORMs to build a FORM for the category ⟨rule⟩. The final rule constructed, then, resides in the FORM attribute of ⟨rule⟩ and is passed on to Prolog as

rule(state(led-2,on),status(transistor-17,failed)).

These lexical functional constraints are the sole definition of the Prolog-based internal knowledge base formats. The GLIB grammar itself depends in no way on the choice of Prolog as a target representation. The rules acquired could be as easily compiled into Lisp, or some other format.

INKA combines the semantic acquisition grammar and internal format definitions to build a parser that acquires rules in the grammar and translates them into the proper internal forms. The rules can then be tested by an inference engine running in the background.

Several features make this a convenient interface to use. When typing a rule, menus can be selected to guide the user (Figure 13). The system also supports phrase completion and spelling correction.

PIKA (Pictorial Knowledge Acquisition) is a graphics editor that produces as its output not a simple picture, but a collection of structured graphical objects. The structure is important to facilitate their use for pointing to and cross-referencing components of the diagram. These structured objects can be assembled to support a multi-level display interface.

Figure 14 shows an example of PIKA being used to create a circuit schematic.

A third tool—still in the design stages—is CHEKA (Checker for Knowledge Acquisition). CHEKA accepts a collection of integrity constraints developed during the analysis of a problem and checks new rules being added by INKA for consistency with these constraints. CHEKA will use techniques developed at the Japanese Institute for New Generation Computer Technology (Kitakami, 1984, Miyachi, 1984) that relies on explicit statement of constraints to determine consistency of a particular set of facts and rules.

In a complete development environment to support our methodology, some tools would be general purpose, while others might be specifically tuned to a particular problem world. People using the tools would choose problem specific tools when available and general tools when specific tools did not exist. The important thing is that the collection of tools support a step-by-step approach to knowledge engineering, always providing some way to keep making progress on the problem at hand.

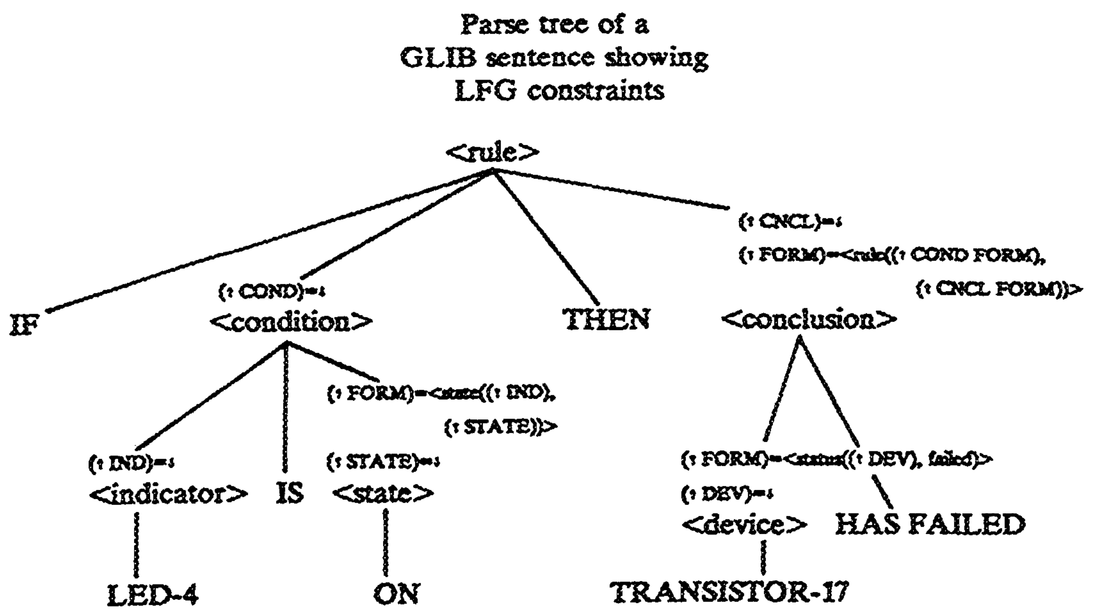
Conclusions

From our experience in teaching the knowledge engineering process to software engineers at Tektronix, we have discovered that the most important need for expert system prototype development is the need for techniques to examine a problem and begin to turn its amorphous shape into something concrete enough so that a prototype system can be created.

We have demonstrated one approach to this problem that uses formal grammars as a documentation technique in managing these early stages of development. Much more work needs to be done, however, to provide a gradual and step-by-step approach that does not require years of training before it can be used. Tools which support these steps of familiarization, definition, and acquisition of an expert's knowledge will doubtless form the backbone of future expert system development tools.

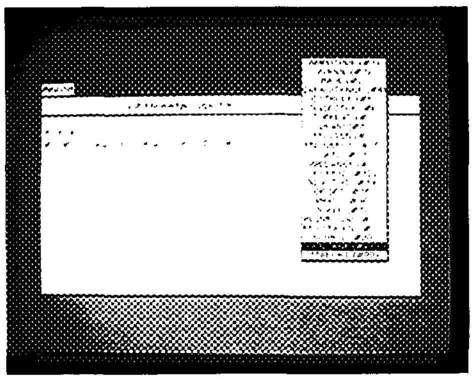
References

Alexander, J H & Freiling, M J. (1985) Smalltalk-80 aids troubleshooting system development *Systems and Software* 4, 4



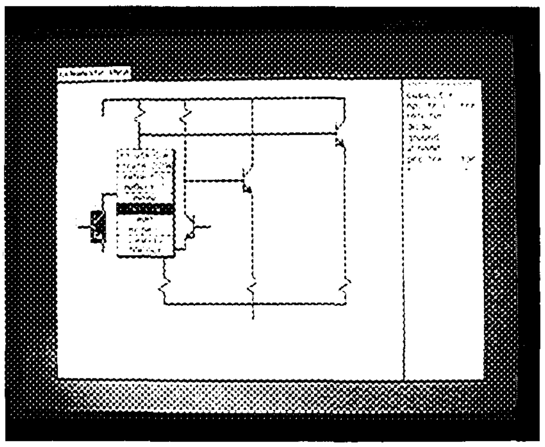
INKA in operation.

Figure 12.



Using menus in INKA to express rules.

Figure 13.



Using PIKA to build a schemataic diagram.

Figure 14.

Alexander, J. H , Freiling, M J., Messick, S L., & Rehfuss, S. (1985a) *Efficient expert system development through domain-specific tools.* Fifth International Workshop on Expert Systems and their Applications.

Allen, J F. (1981) *An interval-based representation of temporal knowledge.* IJCAI-7

Boose, J. H. (1984) *Personal construct theory and the transfer of human expertise* AAAI-84.

Brachman, R. & Levesque, H. (1982) *Competence in knowledge representation.* AAAI-82

Brown, J S., Burton, R. R., & deKleer, J. (1982) Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In D. Sleeman & J S Brown (Eds.), *Intelligent Tutoring Systems* New York: Academic Press

Buchanan, B. G , Barstow, D , Bechtal, R , Bennett, J , Clancey, W., Kulikowski, C , Mitchell, T., & Waterman, D A (1983) Constructing an Expert System In F Hayes-Roth, D. A Waterman, & D B Lenat (Eds.), *Building Expert Systems* Reading, MA: Addison-Wesley

Burton, R R. (1976) Semantic Grammar: An engineering technique for constructing natural language understanding systems Tech. Rep. 3453 Bolt, Beranek, and Newman, Cambridge, MA,

Clancey, W J. (1984) *Classification problem solving* AAAI-84

Feiner, S , Nagy, S , & Dam, A Van (1982) An experimental system for creating and presenting interactive Graphical Documents *ACM Transactions on Graphics* Vol. 1, No 1

HUGHES

MEMBER OF THE TECHNICAL STAFF

Southern California

This is the opportune moment to join Hughes Research Laboratories, in Southern California, and be one of the people behind the development of the most sophisticated electronics technology. At our Malibu facility, we're building high performance VLSI concurrent computing systems for two-dimensional radar, sonar, and computer vision applications. We have immediate openings for Members of the Technical Staff in our Information Sciences and Computer Architecture Section. Here, you'll find professional growth potential due to the wide variety of activities at our facility, such as participating in programs to build sophisticated linear algebraic processor arrays and image understanding systems. New challenge and excitement can be yours at Hughes

A background in one or more of the following areas is required

- Concurrent Symbolic and Numeric Computing Architectures
- Computer Vision
- Knowledge Based Systems
- Inference Machines
- Digital Signal Processing

A Ph D in Electrical Engineering or Computer Science is preferred.

We offer an ideal location, stimulating environment, and excellent employee benefits

Please send your resume and salary history to. Arlyne Luna, Hughes Research Laboratories, Dept. IA-7, 3011 Malibu Canyon Road, Malibu, CA 90265. Proof of U S. Citizenship Required. Equal Opportunity Employer.

HUGHES
AIRCRAFT COMPANY

RESEARCH LABORATORIES

Freiling, M J (1983) SIDUR—an integrated data model IEEE Compcon, IEEE Computer Society

Freiling, M. J & Alexander, J H (1984a) *Diagrams and grammars: tools for the mass production of expert systems.* First Conference on Artificial Intelligence Applications. IEEE Computer Society.

Freiling, M J , Alexander, J. H., Feucht, D , & Stubbs, D (1984) GLIB—a language for describing the behavior of electronic devices Applied Research Tech Rep. CR-84-12 Tektronix, Inc,, Beaverton, OR

Goldberg, A. & Robson, D. (1983) *Smalltalk 80: the language and its implementation.* Reading, MA: Addison-Wesley

Hollan, J D., Hutchins, E. L., & Weitzman, L (1984) STEAMER: an interactive inspectable simulation-based training system. *AI Magazine* Vol 5, No 2.

Kaplan, R. M & Bresnan, J. W. (1982) Lexical-functional grammar: a formal system for grammatical representation. In J. W Bresnan (Ed), *The Mental Representation of Grammatical Relations* Cambridge, MA: MIT Press.

Kitakami, H , Kunifuji, S., Miyachi, T., & Furukawa, K (1984) *A methodology for implementation of a knowledge acquisition system.* 1984 International Symposium on Logic Programming. IEEE Computer Society.

Kogan, D D & Freiling, M. J (1984) *SIDUR—a structuring formalism for knowledge information processing systems.* International Conference on Fifth Generation Computer Systems. Institute for New Generation Computer Technology (ICOT) Tokyo, Japan

Kunz, J., Kehler, T., & Williams, M. (1984) Applications development using a hybrid AI development system *AI Magazine* Vol 5, No. 3

Miyachi, T , Kunifuji, S , Kitakami, H., Furukawa, K , Takeuchi, A , and Yokota, H (1984) *A knowledge assimilation method for logic databases* 1984 International Symposium on Logic Programming. IEEE Computer Society

Phillips, B & Nicholl, S (1984) INGLISH: A natural language interface Applied Research Tech Rep CR-84-27, Tektronix, Inc, Beaverton, OR

Phillips, B., Messick, S L , Freiling, M. J , & Alexander, J H (1985) INKA: The Inglish knowledge acquisition interface for electronic instrument troubleshooting systems. Applied Research Tech Rep CR-85-04, Tektronix, Inc, Beaverton, OR

Stefik, M , Aikins, J , Balzer, R , Benoit, J., Birnbaum, L , Hayes-Roth, F., & Sacerdoti, E. (1982) The organization of expert systems: A prescriptive tutorial. Xerox PARC Tech Rep , Palo Alto, CA.

VanMelle, W , Shortliffe, E. H , & Buchanan, B G. (1984) EMYCIN: A knowledge engineer's tool for constructing rule-based expert systems. In B. G. Buchanan & E H. Shortliffe (Eds), *Rule-Based Expert Systems.* Reading, MA: Addison-Wesley.

Vilain, M. B. (1982) *A system for reasoning about time.* AAAI-82

Employment Opportunity

The Heuristic Programming Project (HPP), a division of the Knowledge Systems Laboratory at Stanford University, is seeking a qualified individual to assist the research efforts of the Advanced Architectures Project. The position will require independent research in parallel programming techniques and especially the implementation of parallel Lisp-like languages. The candidate should have broad experience with Common Lisp or Zetalisp, Symbolics 3600 computers and expert systems. Of particular interest is a knowledge of the implementation of Common Lisp and Scheme-like languages. The position requires a BS degree or higher in Computer Science and at least five years experience in Lisp programming within an AI context For further information, contact:

Dr. Robert Engelmore
Executive Director, HPP Dept. of Computer Science
Stanford University 701 Welch Rd., Bldg. C
Palo Alto, CA 94304 Phone: (415)497-8444
ARPANET: (engelmore@sumex-aim)